0

Notes to the Reader

Writing is the only art
that must be learned by wrote.
– anon

Main themes of this book — how to read this book — a timeline for C++ — C++ and other programming languages — references.

Introduction
C++ was designed to provide Simula's facilities for program organization together with C's efficiency and flexibility for systems programming. It was intended to deliver that to real projects within half a year of the idea. It succeeded.

At the time, mid-1979, neither the modesty nor the preposterousness of that goal was realized. The goal was modest in that it did not involve innovation, and preposterous in both its time scale and its Draconian demands on efficiency and flexibility. While a modest amount of innovation did emerge over the years, efficiency and flexibility have been maintained without compromise. While the goals for C++ have been refined, elaborated, and made more explicit over the years, C++ as used today directly reflects its original aims.

The purpose of this book is to document those aims, track their evolution, and present C++ as it emerged from the efforts of many people to create a language that served its users according to those aims. In doing so, I try to balance historical facts (such as names, places, and events) against technical issues of language design, implementation, and use. It is not my aim to document every little event, but to focus on the key events, ideas, and trends that actually influenced the definition of C++ or might influence its further evolution and use.

Wherever events are presented, I try to describe them as they happened rather than how I or others might have liked them to have happened. Where reasonable, I use quotes from papers to illustrate the aims, principles, and features as they appeared at the time. I try not to project hindsight into events; rather, retrospective comments and comments about the implications of a decision are presented separately and are explicitly marked as retrospective. Basically, I abhor revisionist history and try to avoid it. For example, I mention that ‘‘I had found Pascal’s type system worse than useless – a straitjacket that caused more problems than it solved by forcing me to warp my designs to suit an implementation-oriented artifact.’’ That I thought that at the time is a fact, and it is a fact that had important implications for the evolution of C++. Whether that harsh judgement on Pascal was fair and whether I would make the same judgement today (more than a decade later) is irrelevant. I could not delete the fact (say, to spare the feelings of Pascal fans or to spare myself embarrassment or controversy) or modify it (by providing a more complete and balanced view) without warping the history of C++.

I try to mention people who contributed to the design and evolution of C++, and I try to be specific about their contribution and about when it occurred. This is somewhat hazardous. Since I don’t have a perfect memory, I will overlook some contributions. I offer my apologies. I name the people who caused a decision to be made for C++. Inevitably, these will not always be the people who first encountered a particular problem or who first thought of a solution. This can be unfortunate, but to be vague or to refrain from mentioning names would be worse. Feel free to send me information that might help clarify such points.

Where I describe historical events, there is a question of how objective my descriptions are. I have tried to compensate for unavoidable bias by obtaining information about events I wasn’t part of, by talking to other people involved in events, and by having several of the people involved in the evolution of C++ read this book. Their names can be found at the end of the preface. In addition, the History of Programming Languages (HOPL-2) paper [Stroustrup,1993] that contains the central historical facts from this book was extensively reviewed and deemed free of unsuitable bias.

How to Read this Book

Part I goes through the design, evolution, use, and standardization of C++ in roughly chronological order. I chose this organization because during the early years, major design decisions map onto the timeline as a neat, logical sequence. Chapters 1, 2, and 3 describe the origins of C++ and its evolution through C with Classes to Release 1.0. Chapter 4 describes the rules that guided C++’s growth during that period and beyond. Chapter 5 provides a chronology of post-1.0 developments, and Chapter 6 describes the ANSI/ISO C++ standards effort. To provide perspective, Chapters 7 and 8 discuss applications, tools, and libraries. Finally, Chapter 9 presents a retrospective and some thoughts on the future.

Part II presents the post-Release-1.0 development of C++. The language grew within a framework laid down around the time of Release 1.0. This framework included a set of desired features, such as templates and exception handling, and rules guiding their design. After Release 1.0, chronology didn’t matter much to the development of C++. The current definition of C++ would have been substantially the same had the chronological sequence of post-1.0 extensions been different. The actual sequence in which the problems were solved and features provided is therefore of historical interest only. A strictly chronological presentation would interfere with the logical flow of ideas, so Part II is organized around major language features instead. Part II chapters are independent, so they can be read in any order: Chapter 10, memory management; Chapter 11, overloading; Chapter 12, multiple inheritance; Chapter 13, class concept refinements; Chapter 14, casting; Chapter 15, templates; Chapter 16, exception handling; Chapter 17, namespaces; Chapter 18, the C preprocessor.

Different people expect radically different things from a book on the design and evolution of a programming language. In particular, no two people seem to agree on what level of detail is appropriate for a discussion of this topic. *Every* review I received on the various versions of the HOPL-2 paper (well over a dozen reviews) was of the form, ‘‘This paper is too long ... please add information on topics X, Y, and Z.’’ Worse, about a third of the reviews had comments of the form, ‘‘Cut the philosophical/religious nonsense and give us proper technical details instead.’’ Another third commented, ‘‘Spare me the boring details and add information on your design philosophy.’’

To wiggle out of this dilemma, I have written a book within a book. If you are not interested in details, then at first skip all subsections (numbered §*x.y.z*, where *x* is the chapter number and *y* is the section number). Later, read whatever else looks interesting. You can also read this book sequentially starting at page one and carry on until the end. Doing that, you might get bogged down in details. This is not meant to imply that details are unimportant. On the contrary, no programming language can be understood by considering principles and generalizations only; concrete examples are essential. However, looking at the details without an overall picture to fit them into is a way of getting seriously lost.

As an additional help, I have concentrated most of the discussion of new features and features generally considered advanced in Part II. This allows Part I to concentrate on basics. Almost all of the information on nontechnical aspects of C++'s evolution is found in Part I. People with little patience for ‘‘philosophy’’ can break up the discussion in Chapters 4 through 9 by looking ahead to the technical details of language features in Part II.

I assume that some will use this book as a reference and that many will read individual chapters without bothering with all preceding chapters. To make such use feasible, I have made the individual chapters relatively self-contained for the experienced C++ programmer and been liberal with cross references and index terms.

Please note that I don't try to define the features of C++ here, I present only as much detail as is necessary to provide a self-contained description of how the features came about. I don't try to teach C++ programming or design either; for a tutorial, see [2nd].

C++ Timeline

This C++ timeline might help you keep track of where the story is taking you:

1979	May	Work on C with Classes starts
	Oct	1st C with Classes implementation in use
1980	Apr	1st internal Bell Labs paper on C with Classes [Stroustrup,1980]
1982	Jan	1st external paper on C with Classes [Stroustrup,1982]
1983	Aug	1st C++ implementation in use
	Dec	C++ named
1984	Jan	1st C++ manual
1985	Feb	1st external C++ release (Release E)
	Oct	Cfront Release 1.0 (first commercial release)
	Oct	*The C++ Programming Language* [Stroustrup,1986]
1986	Aug	The ‘‘whatis paper’’ [Stroustrup,1986b]
	Sep	1st OOPSLA conference (start of OO hype centered on Smalltalk)
	Nov	1st commercial Cfront PC port (Cfront 1.1, Glockenspiel)
1987	Feb	Cfront Release 1.2
	Nov	1st USENIX C++ conference (Santa Fe, NM)
	Dec	1st GNU C++ release (1.13)
1988	Jan	1st Oregon Software C++ release
	June	1st Zortech C++ release
	Oct	1st USENIX C++ implementers workshop (Estes Park, CO)
1989	June	Cfront Release 2.0
	Dec	ANSI X3J16 organizational meeting (Washington, DC)
1990	May	1st Borland C++ release
	Mar	1st ANSI X3J16 technical meeting (Somerset, NJ)
	May	*The Annotated C++ Reference Manual* [ARM]
	July	Templates accepted (Seattle, WA)
	Nov	Exceptions accepted (Palo Alto, CA)
1991	June	*The C++ Programming Language (second edition)* [2nd]
	June	1st ISO WG21 meeting (Lund, Sweden)
	Oct	Cfront Release 3.0 (including templates)
1992	Feb	1st DEC C++ release (including templates and exceptions)
	Mar	1st Microsoft C++ release
	May	1st IBM C++ release (including templates and exceptions)
1993	Mar	Run-time type identification accepted (Portland, OR)
	July	Namespaces accepted (Munich, Germany)
1994	Aug	ANSI/ISO Committee Draft registered

Focus on Use and Users

This book is written for C++ users, that is, for programmers and designers. I have tried (believe it or not) to avoid truly obscure and esoteric topics to give a user’s view of the C++ language, its facilities, and its evolution. Purely language-technical discussions are presented only if they shed light on issues that directly impact users. The discussions of name lookup in templates (§15.10) and of lifetime of temporaries (§6.3.2) are examples.

Programming language specialists, language lawyers, and implementers will find many tidbits in this book, but the aim is to present the large picture rather than to be precise and comprehensive about every little detail. If precise language-technical details is what you want the definition of C++ can be found in *The Annotated C++ Reference Manual* (the ARM) [ARM], in *The C++ Programming Language (second edition)* [2nd], and in the ANSI/ISO standards committee's working paper. However, the details of a language definition are incomprehensible without an understanding of the purpose of the language. The language, details and all, exists to help build programs. My intent with this book is to provide insights that can help in this endeavor.

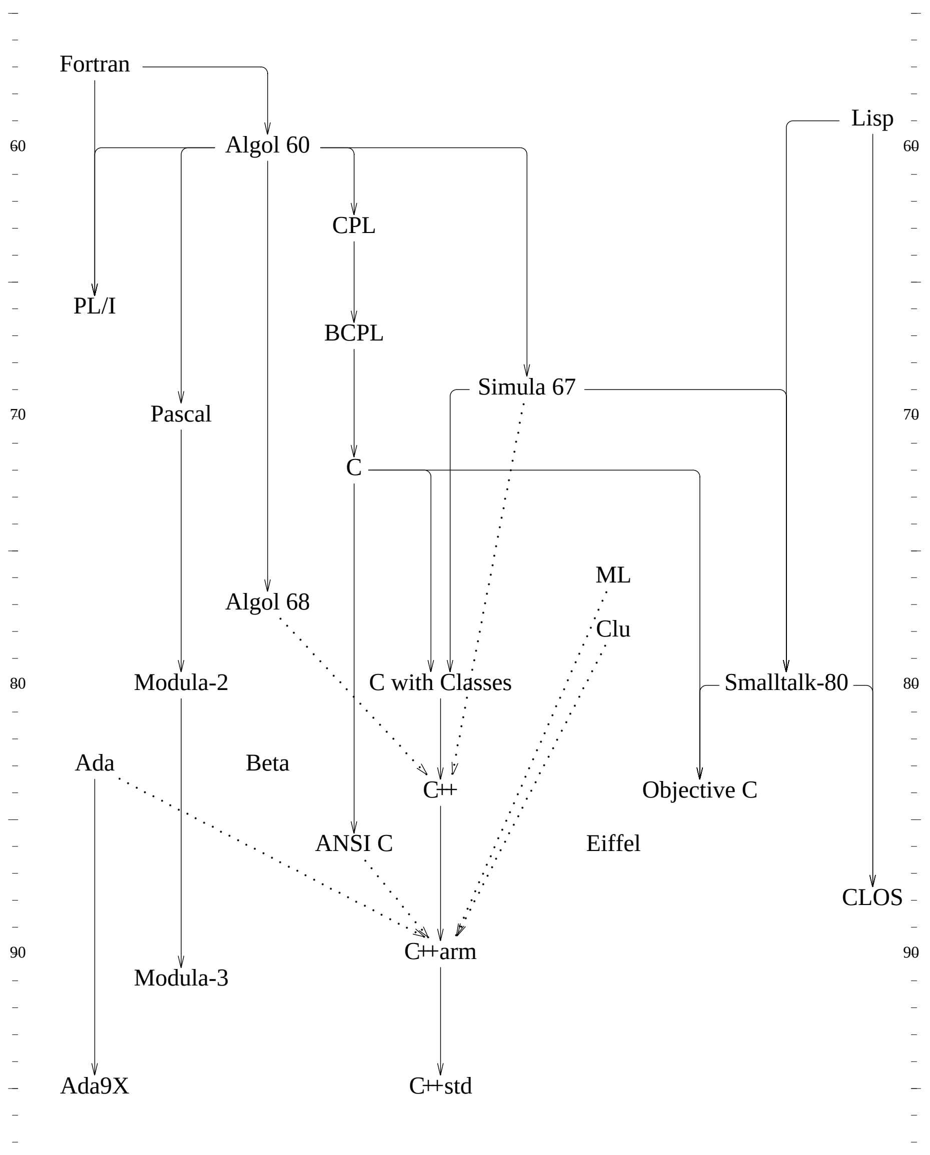
Programming Languages

Several reviewers asked me to compare C++ to other languages. This I have decided against doing. Thereby, I have reaffirmed a long-standing and strongly held view: Language comparisons are rarely meaningful and even less often fair. A good comparison of major programming languages requires more effort than most people are willing to spend, experience in a wide range of application areas, a rigid maintenance of a detached and impartial point of view, and a sense of fairness. I do not have the time, and as the designer of C++, my impartiality would never be fully credible.

I also worry about a phenomenon I have repeatedly observed in honest attempts at language comparisons. The authors try hard to be impartial, but are hopelessly biased by focusing on a single application, a single style of programming, or a single culture among programmers. Worse, when one language is significantly better known than others, a subtle shift in perspective occurs: Flaws in the well-known language are deemed minor and simple workarounds are presented, whereas similar flaws in other languages are deemed fundamental. Often, the workarounds commonly used in the less-well-known languages are simply unknown to the people doing the comparison or deemed unsatisfactory because they would be unworkable in the more familiar language.

Similarly, information about the well-known language tends to be completely up-to-date, whereas for the less-known language, the authors rely on several-year-old information. For languages that are worth comparing, a comparison of language X as defined three years ago vs. language Y as it appears in the latest experimental implementation is neither fair nor informative. Thus, I restrict my comments about languages other than C++ to generalities and to very specific comments. This is a book about C++, its design, and the factors that shaped its evolution. It is not an attempt to contrast C++ language features with those found in other languages.

To fit C++ into a historical context, here is a chart of the first appearances of languages that often crop up in discussions about C++:



The chart is not intended to be anywhere near complete except for significant influences on C++. In particular, the chart understates the influence of the Simula class concept; Ada [Ichbiah,1979] and Clu [Liskov,1979] are weakly influenced by Simula [Birtwistle,1979]; Ada9X [Taft,1992], Beta [Madsen,1993], Eiffel [Meyer,1988], and Modula-3 [Nelson,1991] are strongly influenced. C++'s influence on other languages is left unrepresented. Solid lines indicate an influence on the structure of the language; dotted lines indicate an influence on specific features. Adding lines to show this for every language would make the diagram too messy to be useful. The dates for the languages are generally those of the first usable implementation. For example, Algol68 [Woodward,1974] can be found by the year 1977 rather than 1968.

One conclusion I drew from the wildly divergent comments on the HOPL-2 paper – and from many other sources – is that there is no agreement on what a programming language really is and what its main purpose is supposed to be. Is a programming language a tool for instructing machines? A means of communicating between programmers? A vehicle for expressing high-level designs? A notation for algorithms? A way of expressing relationships between concepts? A tool for experimentation? A means of controlling computerized devices? My view is that a general-purpose programming language must be all of those to serve its diverse set of users. The only thing a language cannot be – and survive – is a mere collection of ''neat'' features.

The difference in opinions reflects differing views of what computer science is and how languages ought to be designed. Ought computer science be a branch of mathematics? Of engineering? Of architecture? Of art? Of biology? Of sociology? Of philosophy? Alternatively, does it borrow techniques and approaches from all of these disciplines? I think so.

This implies that language design parts ways from the ''purer'' and more abstract disciplines such as mathematics and philosophy. To serve its users, a general-purpose programming language must be eclectic and take many practical and sociological factors into account. In particular, every language is designed to solve a particular set of problems at a particular time according to the understanding of a particular group of people. From this initial design, it grows to meet new demands and reflects new understandings of problems and of tools and techniques for solving them. This view is pragmatic, yet not unprincipled. It is my firm belief that *all* successful languages are grown and not merely designed from first principles. Principles underlie the first design and guide the further evolution of the language. However, even principles evolve.

References

This section contains the references from every chapter of this book.

[2nd]	see [Stroustrup,1991].
[Agha,1986]	Gul Agha: *An Overview of Actor languages*. ACM SIGPLAN Notices. October 1986.
[Aho,1986]	Alfred Aho, Ravi Sethi, and Jeffrey D. Ullman: *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA. 1986. ISBN 0-201-10088-6.
[ARM]	see [Ellis,1990].
[Babcisky,1984]	Karel Babcisky: *Simula Performance Assessment*. Proc. IFIP WG2.4 Conference on System Implementation Languages: Experience and Assessment. Canterbury, Kent, UK. September 1984.
[Barton,1994]	John J. Barton and Lee R. Nackman: *Scientific and*

Engineering C++: An Introduction with Advanced Techniques and Examples. Addison-Wesley, Reading, MA. 1994. ISBN 0-201-53393-6.

[Birtwistle,1979] Graham Birtwistle, Ole-Johan Dahl, Björn Myrhaug, and Kristen Nygaard: *SIMULA BEGIN*. Studentlitteratur, Lund, Sweden. 1979. ISBN 91-44-06212-5.

[Boehm,1993] Hans-J. Boehm: *Space Efficient Conservative Garbage Collection*. Proc. ACM SIGPLAN '93 Conference on Programming Language Design and Implementation. ACM SIGPLAN Notices. June 1993.

[Booch,1990] Grady Booch and Michael M. Vilot: *The Design of the C++ Booch Components*. Proc. OOPSLA'90. October 1990.

[Booch,1991] Grady Booch: *Object-Oriented Design*. Benjamin Cummings, Redwood City, CA. 1991. ISBN 0-8053-0091-0.

[Booch,1993] Grady Booch: *Object-oriented Analysis and Design with Applications, 2nd edition*. Benjamin Cummings, Redwood City, CA. 1993. ISBN 0-8053-5340-2.

[Booch,1993b] Grady Booch and Michael M. Vilot: *Simplifying the C++ Booch Components*. The C++ Report. June 1993.

[Budge,1992] Ken Budge, J.S. Perry, and A.C. Robinson: *High-Performance Scientific Computation using C++*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Buhr,1992] Peter A. Buhr and Glen Ditchfield: *Adding Concurrency to a Programming Language*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Call,1987] Lisa A. Call, et al.: *CLAM – An Open System for Graphical User Interfaces*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Cameron,1992] Don Cameron, et al.: *A Portable Implementation of C++ Exception Handling*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Campbell,1987] Roy Campbell, et al.: *The Design of a Multiprocessor Operating System*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Cattell,1991] Rich G.G. Cattell: *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Addison-Wesley, Reading, MA. 1991. ISBN 0-201-53092-9.

[Cargill,1991] Tom A. Cargill: *The Case Against Multiple Inheritance in C++*. USENIX Computer Systems. Vol 4, no 1, 1991.

[Carroll,1991] Martin Carroll: *Using Multiple Inheritance to Implement Abstract Data Types*. The C++ Report. April 1991.

[Carroll,1993] Martin Carroll: *Design of the USL Standard Components*. The C++ Report. June 1993.

[Chandy,1993] K. Mani Chandy and Carl Kesselman: *Compositional C++:*

Compositional Parallel Programming. Proc. Fourth Workshop on Parallel Computing and Compilers. Springer-Verlag. 1993.

[Cristian,1989] Flaviu Cristian: *Exception Handling*. Dependability of Resilient Computers, T. Andersen, editor. BSP Professional Books, Blackwell Scientific Publications, 1989.

[Cox,1986] Brad Cox: *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, Reading, MA. 1986.

[Dahl,1988] Ole-Johan Dahl: Personal communication.

[Dearle,1990] Fergal Dearle: *Designing Portable Applications Frameworks for C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.

[Dorward,1990] Sean M. Dorward, et al.: *Adding New Code to a Running Program*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.

[Eick,1991] Stephen G. Eick: *SIMLIB - An Object-Oriented C++ Library for Interactive Simulation of Circuit-Switched Networks*. Proc. Simulation Technology Conference. Orlando, FL. October 1991.

[Ellis,1990] Margaret A. Ellis and Bjarne Stroustrup: *The Annotated C++ Reference Manual*. Addison-Wesley, Reading, MA. 1990. ISBN 0-201-51459-1.

[Faust,1990] John E. Faust and Henry M. Levy: *The Performance of an Object-Oriented Threads Package*. Proc. ACM joint ECOOP and OOPSLA Conference. Ottawa, Canada. October 1990.

[Fontana,1991] Mary Fontana and Martin Neath: *Checked Out and Long Overdue: Experiences in the Design of a C++ Class Library*. Proc. USENIX C++ Conference. Washington, DC. April 1991.

[Forslund,1990] David W. Forslund, et al.: *Experiences in Writing Distributed Particle Simulation Code in C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.

[Gautron,1992] Philippe Gautron: *An Assertion Mechanism based on Exceptions*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Gehani,1988] Narain H. Gehani and William D. Roome: *Concurrent C++: Concurrent Programming With Class(es)*. Software—Practice & Experience. Vol 18, no 12, 1988.

[Goldberg,1983] Adele Goldberg and David Robson: *Smalltalk-80, The Language and its Implementation*. Addison-Wesley, Reading, MA. 1983. ISBN 0-201-11371-6.

[Goodenough,1975] John Goodenough: *Exception Handling: Issues and a Proposed Notation*. Communications of the ACM. December 1975.

[Gorlen,1987] Keith E. Gorlen: *An Object-Oriented Class Library for C++ Programs*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Gorlen,1990] Keith E. Gorlen, Sanford M. Orlow, and Perry S. Plexico: *Data Abstraction and Object-Oriented Programming in C++*. Wiley. West Sussex. England. 1990. ISBN 0-471-92346-X.

[Hübel,1992] Peter Hübel and J.T. Thorsen: *An Implementation of a Persistent Store for C++*. Computer Science Department. Aarhus University, Denmark. December 1992.

[Ichbiah,1979] Jean D. Ichbiah, et al.: *Rationale for the Design of the ADA Programming Language*. SIGPLAN Notices Vol 14, no 6, June 1979 Part B.

[Ingalls,1986] Daniel H.H. Ingalls: *A Simple Technique for Handling Multiple Polymorphism*. Proc. ACM OOPSLA Conference. Portland, OR. November 1986.

[Interrante,1990] John A. Interrante and Mark A. Linton: *Runtime Access to Type Information*. Proc. USENIX C++ Conference. San Francisco 1990.

[Johnson,1992] Steve C. Johnson: Personal communication.

[Johnson,1989] Ralph E. Johnson: *The Importance of Being Abstract*. The C++ Report. March 1989.

[Keffer,1992] Thomas Keffer: *Why C++ Will Replace Fortran*. C++ Supplement to Dr. Dobbs Journal. December 1992.

[Keffer,1993] Thomas Keffer: *The Design and Architecture of Tools.h++*. The C++ Report. June 1993.

[Kernighan,1976] Brian Kernighan and P.J. Plauger: *Software Tools*. Addison-Wesley, Reading, MA. 1976. ISBN 0-201-03669.

[Kernighan,1978] Brian Kernighan and Dennis Ritchie: *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ. 1978. ISBN 0-13-110163-3.

[Kernighan,1981] Brian Kernighan: *Why Pascal is not my Favorite Programming Language*. AT&T Bell Labs Computer Science Technical Report No 100. July 1981.

[Kernighan,1984] Brian Kernighan and Rob Pike: *The UNIX Programming Environment*. Prentice-Hall, Englewood Cliffs, NJ. 1984. ISBN 0-13-937699-2.

[Kernighan,1988] Brian Kernighan and Dennis Ritchie: *The C Programming Language (second edition)*. Prentice-Hall, Englewood Cliffs, NJ. 1988. ISBN 0-13-110362-8.

[Kiczales,1992] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow: *The Art of the Metaobject Protocol*. The MIT Press. Cambridge, Massachusetts. 1991. ISBN 0-262-11158-6.

[Koenig,1988] Andrew Koenig: *Associative arrays in C++*. Proc. USENIX Conference. San Francisco, CA. June 1988.

[Koenig,1989] Andrew Koenig and Bjarne Stroustrup: *C++: As close to C as possible – but no closer*. The C++ Report. July 1989.

[Koenig,1989b] Andrew Koenig and Bjarne Stroustrup: *Exception Handling for*

C++. Proc. ‘‘C++ at Work’’ Conference. November 1989.

[Koenig,1990] Andrew Koenig and Bjarne Stroustrup: *Exception Handling for C++ (revised)*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990. Also, Journal of Object-Oriented Programming. July 1990.

[Koenig,1991] Andrew Koenig: *Applicators, Manipulators, and Function Objects*. C++ Journal, vol. 1, #1. Summer 1990.

[Koenig,1992] Andrew Koenig: *Space Efficient Trees in C++*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Krogdahl,1984] Stein Krogdahl: *An Efficient Implementation of Simula Classes with Multiple Prefixing*. Research Report No 83. June 1984. University of Oslo, Institute of Informatics.

[Lea,1990] Doug Lea and Marshall P. Cline: *The Behavior of C++ Classes*. Proc. ACM SOOPPA Conference. September 1990.

[Lea,1991] Doug Lea: Personal Communication.

[Lea,1993] Doug Lea: *The GNU C++ Library*. The C++ Report. June 1993.

[Lenkov,1989] Dmitry Lenkov: *C++ Standardization Proposal*. #X3J11/89-016.

[Lenkov,1991] Dmitry Lenkov, Michey Mehta, and Shankar Unni: *Type Identification in C++*. Proc. USENIX C++ Conference. Washington, DC. April 1991.

[Linton,1987] Mark A. Linton and Paul R. Calder: *The Design and Implementation of InterViews*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Lippman,1988] Stan Lippman and Bjarne Stroustrup: *Pointers to Class Members in C++*. Proc. USENIX C++ Conference. Denver, CO. October 1988.

[Liskov,1979] Barbara Liskov, et al.: *CLU Reference manual*. MIT/LCS/TR-225. October 1979.

[Liskov,1987] Barbara Liskov: *Data Abstraction and Hierarchy*. Addendum to Proceedings of OOPSLA’87. October 1987.

[Madsen,1993] Ole Lehrmann Madsen, et al.: *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley, Reading, MA. 1993. ISBN 0-201-62430.

[McCluskey,1992] Glen McCluskey: *An Environment for Template Instantiation*. The C++ Report. February 1992.

[Meyer,1988] Bertrand Meyer: *Object-Oriented Software Construction*. Prentice-Hall, Englewood Cliffs, NJ. 1988. ISBN 0-13-629049.

[Miller,1988] William M. Miller: *Exception Handling without Language Extensions*. Proc. USENIX C++ Conference. Denver CO. October 1988.

[Mitchell,1979] James G. Mitchell, et.al.: *Mesa Language Manual*. XEROX

PARC, Palo Alto, CA. CSL-79-3. April 1979.

[Murray,1992] Rob Murray: *A Statically Typed Abstract Representation for C++ Programs*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Nelson,1991] Nelson, G. (editor): *Systems Programming with Modula-3*. Prentice-Hall, Englewood Cliffs, NJ. 1991. ISBN 0-13-590464-1.

[Rose,1984] Leonie V. Rose and Bjarne Stroustrup: *Complex Arithmetic in C++*. Internal AT&T Bell Labs Technical Memorandum. January 1984. Reprinted in AT&T C++ Translator Release Notes. November 1985.

[Parrington,1990] Graham D. Parrington: *Reliable Distributed Programming in C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.

[Reiser,1992] John F. Reiser: *Static Initializers: Reducing the Value-Added Tax on Programs*. Proc. USENIX C++ Conference. Portland, OR. August 1992.

[Richards,1980] Martin Richards and Colin Whitby-Strevens: *BCPL – the language and its compiler*. Cambridge University Press, Cambridge, England. 1980. ISBN 0-521-21965-5.

[Rovner,1986] Paul Rovner: *Extending Modula-2 to Build Large, Integrated Systems.* IEEE Software Vol 3, No 6, November 1986.

[Russo,1988] Vincent F. Russo and Simon M. Kaplan: *A C++ Interpreter for Scheme*. Proc. USENIX C++ Conference. Denver, CO. October 1988.

[Russo,1990] Vincent F. Russo, Peter W. Madany, and Roy H. Campbell: *C++ and Operating Systems Performance: A Case Study*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.

[Sakkinen,1992] Markku Sakkinen: *A Critique of the Inheritance Principles of C++*. USENIX Computer Systems, vol 5, no 1, Winter 1992.

[Sethi,1980] Ravi Sethi: *A case study in specifying the semantics of a programming language*. Seventh Annual ACM Symposium on Principles of Programming Languages. January 1980.

[Sethi,1981] Ravi Sethi: *Uniform Syntax for Type Expressions and Declarators*. Software – Practice and Experience, Vol 11. 1981.

[Sethi,1989] Ravi Sethi: *Programming Languages – Concepts and Constructs*. Addison-Wesley, Reading, MA. 1989. ISBN 0-201-10365-6.

[Shopiro,1985] Jonathan E. Shopiro: *Strings and Lists for C++*. AT&T Bell Labs Internal Technical Memorandum. July 1985.

[Shopiro,1987] Jonathan E. Shopiro: *Extending the C++ Task System for Real-Time Control*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Shopiro,1989]	Jonathan E. Shopiro: *An Example of Multiple Inheritance in C++: A Model of the Iostream Library*. ACM SIGPLAN Notices. December 1989.
[Schwarz,1989]	Jerry Schwarz: *Iostreams Examples*. AT&T C++ Translator Release Notes. June 1989.
[Snyder,1986]	Alan Snyder: *Encapsulation and Inheritance in Object-Oriented Programming Languages*. Proc. OOPSLA'86. September 1986.
[Stal,1993]	Michael Stal and Uwe Steinmüller: *Generic Dynamic Arrays*. The C++ Report. October 1993.
[Stepanov,1993]	Alexander Stepanov and David R. Musser: *Algorithm-Oriented Generic Software Library Development*. HP Laboratories Technical Report HPL-92-65. November 1993.
[Stroustrup,1978]	Bjarne Stroustrup: *On Unifying Module Interfaces*. ACM Operating Systems Review Vol 12 No 1. January 1978.
[Stroustrup,1979]	Bjarne Stroustrup: *Communication and Control in Distributed Computer Systems*. Ph.D. thesis, Cambridge University, 1979.
[Stroustrup,1979b]	Bjarne Stroustrup: *An Inter-Module Communication System for a Distributed Computer System*. Proc. 1st International Conf. on Distributed Computing Systems. October 1979.
[Stroustrup,1980]	Bjarne Stroustrup: *Classes: An Abstract Data Type Facility for the C Language*. Bell Laboratories Computer Science Technical Report CSTR-84. April 1980. Revised, August 1981. Revised yet again and published as [Stroustrup,1982].
[Stroustrup,1980b]	Bjarne Stroustrup: *A Set of C Classes for Co-routine Style Programming*. Bell Laboratories Computer Science Technical Report CSTR-90. November 1980.
[Stroustrup,1981]	Bjarne Stroustrup: *Long Return: A Technique for Improving The Efficiency of Inter-Module Communication*. Software Practice and Experience. January 1981.
[Stroustrup,1981b]	Bjarne Stroustrup: *Extensions of the C Language Type Concept*. Bell Labs Internal Memorandum. January 1981.
[Stroustrup,1982]	Bjarne Stroustrup: *Classes: An Abstract Data Type Facility for the C Language*. ACM SIGPLAN Notices. January 1982. Revised version of [Stroustrup,1980].
[Stroustrup,1982b]	Bjarne Stroustrup: *Adding Classes to C: An Exercise in Language Evolution*. Bell Laboratories Computer Science internal document. April 1982. Software: Practice & Experience, Vol 13. 1983.
[Stroustrup,1984]	Bjarne Stroustrup: *The C++ Reference Manual*. AT&T Bell Labs Computer Science Technical Report No 108. January 1984. Revised, November 1984.
[Stroustrup,1984b]	Bjarne Stroustrup: *Operator Overloading in C++*. Proc. IFIP WG2.4 Conference on System Implementation Languages:

Experience & Assessment. September 1984.

[Stroustrup,1984c] Bjarne Stroustrup: *Data Abstraction in C*. Bell Labs Technical Journal. Vol 63, No 8. October 1984.

[Stroustrup,1985] Bjarne Stroustrup: *An Extensible I/O Facility for C++*. Proc. Summer 1985 USENIX Conference. June 1985.

[Stroustrup,1986] Bjarne Stroustrup: *The C++ Programming Language*. Addison-Wesley, Reading, MA. 1986. ISBN 0-201-12078-X.

[Stroustrup,1986b] Bjarne Stroustrup: *What is Object-Oriented Programming?* Proc. 14th ASU Conference. August 1986. Revised version in Proc. ECOOP'87, May 1987, Springer Verlag Lecture Notes in Computer Science Vol 276. Revised version in *IEEE Software Magazine*. May 1988.

[Stroustrup,1986c] Bjarne Stroustrup: *An Overview of C++*. ACM SIGPLAN Notices. October 1986.

[Stroustrup,1987] Bjarne Stroustrup: *Multiple Inheritance for C++*. Proc. EUUG Spring Conference, May 1987. Also, USENIX Computer Systems, Vol 2 No 4. Fall 1989.

[Stroustrup,1987b] Bjarne Stroustrup and Jonathan Shopiro: *A Set of C classes for Co-Routine Style Programming.* Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Stroustrup,1987c] Bjarne Stroustrup: *The Evolution of C++: 1985-1987*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Stroustrup,1987d] Bjarne Stroustrup: *Possible Directions for C++*. Proc. USENIX C++ Conference. Santa Fe, NM. November 1987.

[Stroustrup,1988] Bjarne Stroustrup: *Type-safe Linkage for C++*. USENIX Computer Systems, Vol 1 No 4. Fall 1988.

[Stroustrup,1988b] Bjarne Stroustrup: *Parameterized Types for C++*. Proc. USENIX C++ Conference, Denver, CO. October 1988. Also, USENIX Computer Systems, Vol 2 No 1. Winter 1989.

[Stroustrup,1989] Bjarne Stroustrup: *Standardizing C++*. The C++ Report. Vol 1 No 1. January 1989.

[Stroustrup,1989b] Bjarne Stroustrup: *The Evolution of C++: 1985-1989*. USENIX Computer Systems, Vol 2 No 3. Summer 1989. Revised version of [Stroustrup,1987c].

[Stroustrup,1990] Bjarne Stroustrup: *On Language Wars*. Hotline on Object-Oriented Technology. Vol 1, No 3. January 1990.

[Stroustrup,1990b] Bjarne Stroustrup: *Sixteen Ways to Stack a Cat*. The C++ Report. October 1990.

[Stroustrup,1991] Bjarne Stroustrup: *The C++ Programming Language (2nd edition)*. Addison-Wesley, Reading, MA. 1991. ISBN 0-201-53992-6.

[Stroustrup,1992] Bjarne Stroustrup and Dmitri Lenkov: *Run-Time Type Identification for C++*. The C++ Report. March 1992. Revised version: Proc. USENIX C++ Conference. Portland, OR. August

	1992.
[Stroustrup,1992b]	Bjarne Stroustrup: *How to Write a C++ Language Extension Proposal*. The C++ Report. May 1992.
[Stroustrup,1993]	Bjarne Stroustrup: *The History of C++: 1979-1991*. Proc. ACM History of Programming Languages Conference (HOPL-2). April 1993. ACM SIGPLAN Notices. March 1993.
[Taft,1992]	S. Tucker Taft: *Ada 9X: A Technical Summary*. CACM. November 1992.
[Tiemann,1987]	Michael Tiemann: *‘‘Wrappers:’’ Solving the RPC problem in GNU C++*. Proc. USENIX C++ Conference. Denver, CO. October 1988.
[Tiemann,1990]	Michael Tiemann: *An Exception Handling Implementation for C++*. Proc. USENIX C++ Conference. San Francisco, CA. April 1990.
[Weinand,1988]	Andre Weinand, et al.: *ET++ – An Object-Oriented Application Framework in C++*. Proc. OOPSLA’88. September 1988.
[Wikström,1987]	Åke Wikström: *Functional Programming in Standard ML*. Prentice-Hall, Englewood Cliffs, NJ. 1987. ISBN 0-13-331968-7.
[Waldo,1991]	Jim Waldo: *Controversy: The Case for Multiple Inheritance in C++*. USENIX Computer Systems, vol 4, no 2, Spring 1991.
[Waldo,1993]	Jim Waldo (editor): *The Evolution of C++*. A USENIX Association book. The MIT Press, Cambridge, MA. 1993. ISBN 0-262-73107-X.
[Wilkes,1979]	M.V. Wilkes and R.M. Needham: *The Cambridge CAP Computer and its Operating System*. North-Holland, New York. 1979. ISBN 0-444-00357-6.
[Woodward,1974]	P.M. Woodward and S.G. Bond: *Algol 68-R Users Guide*. Her Majesty’s Stationery Office, London. 1974. ISBN 0-11-771600-6.